Available online at www.sciencedirect.com

SCIENCE DIRECT®

ELSEVIER

Information Systems 30 (2005) 227–244

www.elsevier.com/locate/infosys

Sliding window filtering: an efficient method for incremental mining on a time-variant database[☆]

Chang-Hung Lee, Cheng-Ru Lin, Ming-Syan Chen*

Department of Electrical Engineering, National Taiwan University, No. 1, Sec. 4, Roosevelt Rd., Taipei, Taiwan, ROC

Received 3 April 2003; accepted 4 February 2004

Abstract

Recently, several important database applications have called for the design of efficient techniques for incremental mining of association rules. In response to this need, we explore in this paper an effective sliding-window filtering (abbreviatedly as SWF) algorithm for incremental mining of association rules. In essence, by partitioning a transaction database into several partitions, algorithm SWF employs a filtering threshold in each partition to deal with the candidate itemset generation. Under SWF, the cumulative information of mining previous partitions is selectively carried over toward the generation of candidate itemsets for the subsequent partitions. Algorithm SWF not only significantly reduces I/O and CPU cost by the concepts of cumulative filtering and scan reduction techniques but also effectively controls memory utilization by the technique of sliding-window partition. More importantly, algorithm SWF is particularly powerful for efficient incremental mining for an ongoing time-variant transaction database. By utilizing proper scan reduction techniques, only one scan of the incremented dataset is needed by algorithm SWF. The I/O cost of SWF is, in orders of magnitude, smaller than those required by prior methods, thus resolving the performance bottleneck. Extensive experimental studies are performed to evaluate performance of algorithm SWF. Sensitivity analysis of various parameters is conducted to provide many insights into algorithm SWF. It is noted that the improvement achieved by algorithm SWF is even more prominent as the incremented portion of the dataset increases and also as the size of the database increases.
© 2004 Elsevier Ltd. All rights reserved.

Keywords: Data mining; Association rules; Sliding-window filtering; Incremental mining; Time-variant database

1. Introduction

Due to the increasing use of computing for various applications, the importance of data mining is growing at rapid pace. It is noted that analysis of past transaction data can provide valuable information on customer buying behavior, and thus improve the quality of business decisions. In essence, it is necessary to collect and analyze a sufficient amount of sales data before any meaningful conclusion can be drawn

[☆]Partial results of this study appeared in the Proceedings of the 10th ACM International Conference on Information and Knowledge Management (CIKM), November 5–10, 2001.

*Corresponding author. Tel.: +886-2-2363-5251; fax: +886-2-2367-1597.

E-mail addresses: chlee@arbor.ee.ntu.edu (C.-H. Lee), owenlin@arbor.ee.ntu.edu.tu (C.-R. Lin), mschen@cc.ee.ntu.edu.tw (M.-S. Chen).

therefrom. Since the amount of these processed data tends to be huge, it is important to devise efficient algorithms to conduct mining on these data. Various data mining capabilities have been explored in Refs. [1–10]. One receiving a significant amount of research attention is on mining association rules over basket data [1,11–19]. For example, given a database of sales transactions, it is desirable to discover all associations among items such that the presence of some items in a transaction will imply the presence of other items in the same transaction, e.g., 90% of customers that purchase milk and bread also purchase eggs at the same time.

Mining association rules was first introduced in [1], where it was shown that the problem of mining association rules is composed of the following two subproblems: (1) discovering the frequent itemsets, i.e., all sets of itemsets that have transaction support above a pre-determined minimum support s, and (2) using the frequent itemsets to generate the association rules for the database. The overall performance of mining association rules is in fact determined by the first subproblem. After the frequent itemsets are identified, the corresponding association rules can be derived in a straightforward manner [1]. Among others, Apriori [1], DHP [17], and partition-based ones [18,20] are proposed to solve the first subproblem efficiently. In addition, several novel mining techniques, including TreeProjection [21], FP-tree [14,22–24], and constraint-based ones [19,25–29] also received a significant amount of research attention.

In addition, it is noted that recent important applications have called for the need of incremental mining. This is due to the increasing use of the record-based databases whose data are being continuously added. Examples of such applications include Web log records, stock market data, grocery sales data, transactions in electronic commerce, and daily weather/traffic records, to name a few. In many applications, we would like to mine the transaction database for a fixed amount of most recent data (say, data in the last 12 months). That is, in the incremental mining, one has to not only include new data (i.e., data in the new month) into, but also remove the old data (i.e., data in the most obsolete month) from the mining process.

Consider the example transaction database in Fig. 1. Note that $db^{i,j}$ is the part of the transaction database formed by a continuous region from partition P_i to partition P_j. Suppose, we have conducted the mining for the transaction database $db^{i,j}$. As time advances, we are given the new data of January of 2001, and are interested in conducting an incremental mining against the new data. Instead of taking all the past data into consideration, our interest is limited to mining the data in the last 12 months. As a result, the mining of the transaction database $db^{i+1,j+1}$ is called for. Note that since the underlying transaction database has been changed as time advances, some algorithms, such as Apriori, may have to resort to the regeneration of candidate itemsets for the determination of new frequent itemsets, which is, however, very costly even if the incremental data subset is small. On the other hand, while FP-tree-based methods [14,22–24] are shown to be efficient for small databases, it is expected that their deficiency of memory overhead due to the need of keeping a portion of database in memory, as indicated in [30], could become more severe in the presence of a large database upon which an incremental mining process is usually performed.

To the best of our knowledge, there is little progress made thus far to explicitly address the problem of incremental mining except noted below. In [31], the FUP algorithm updates the association rules in a database when new transactions are *added* to the database. Algorithm FUP is based on the framework of Apriori and is designed to discover the new frequent itemsets iteratively.

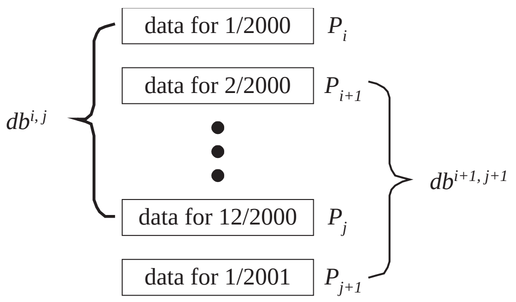


Fig. 1. Incremental mining for an ongoing time-variant transaction database.

The idea is to store the counts of all the frequent itemsets found in a previous mining operation. Using these stored counts and examining the newly added transactions, the overall count of these candidate itemsets are then obtained by scanning the original database. An extension to the work in [31] was reported in [32] where the authors propose an algorithm FUP_2 for updating the existing association rules when transactions are *added* to and *deleted* from the database. In essence, FUP_2 is equivalent to FUP for the case of insertion, and is, however, a complementary algorithm of FUP for the case of deletion. It is shown in [32] that FUP_2 outperforms Apriori algorithm which, without any provision for incremental mining, has to re-run the association rule mining algorithm on the whole updated database. Another FUP-based algorithm, call $FUP_2\mathcal{H}$, was also devised in [32] to utilize the hash technique for performance improvement. Furthermore, the concept of *negative borders* in [33] and that of UWEP, i.e., update with early pruning, in [34] are utilized to enhance the efficiency of FUP-based algorithms.

However, as will be shown by our experimental results, the above mentioned FUP-based algorithms tend to suffer from two inherent problems, namely (1) the occurrence of a potentially huge set of candidate itemsets, and (2) the need of multiple scans of database. First, consider the problem of a potentially huge set of candidate itemsets. Note that the FUP-based algorithms deal with the combination of two sets of candidate itemsets which are independently generated, i.e., from the original data set and the incremental data subset. Since the set of candidate itemsets includes all the possible permutations of the elements, FUP-based algorithms may suffer from a very large set of candidate itemsets, especially from candidate 2-itemsets. As conformed by our experimental results, this problem becomes even more severe for FUP-based algorithms when the increased portion of the incremental mining is large. More importantly, in many applications, one may encounter new itemsets in the increased dataset. While adding some new products in the transaction database, FUP-based algorithms will need to resort to multiple scans of database. Specifically, in the presence of a new frequent itemset L_k generated in the data subset, k scans of the database are needed by FUP-based algorithms in the worst case. That is, the case of $k = 8$ means that the database has to be scanned 8 times, which is very costly, especially in terms of I/O cost. As will become clear later, the problem of a large set of candidate itemsets will hinder an effective use of the scan reduction technique [17] by an FUP-based algorithm.

To remedy these problems, we shall devise in this paper an algorithm based on sliding-window filtering (abbreviatedly as SWF) for incremental mining of association rules. In essence, by partitioning a transaction database into several partitions, algorithm SWF employs a filtering threshold in each partition to deal with the candidate itemset generation. For ease of exposition, the processing of a partition is termed a *phase* of processing. Under SWF, the cumulative information in the prior phases is selectively carried over toward the generation of candidate itemsets in the subsequent phases. After the processing of a phase, algorithm SWF outputs a *cumulative filter*, denoted by *CF*, which consists of a progressive candidate set of itemsets, their occurrence counts and the corresponding partial support required. As will be seen, the cumulative filter produced in each processing phase constitutes the key component to realize the incremental mining. An illustrative example for the operations of SWF is presented in Section 3.1, a detailed description of algorithm SWF is given in Section 3.2 and the correctness of algorithm SWF is proved in Section 3.3. It will be seen that algorithm SWF proposed has several important advantages. First, with employing the prior knowledge in the previous phase, SWF is able to reduce the amount of candidate itemsets efficiently which in turn reduces the CPU and memory overhead. The second advantage of SWF is that owing to the small number of candidate sets generated, the scan reduction technique [17] can be applied efficiently. As a result, only one scan of the ongoing time-variant database is required. As will be validated by our experimental results, this very advantage of SWF enables SWF to significantly outperform FUP-based algorithms. The third advantage of SWF over FUP-based algorithms is the capability of SWF to avoid the data skew in nature. As mentioned in [18,20], such instances as severe weather conditions may cause the sales of some items to increase rapidly within a short period of time. Data skew may cause FUP-based algorithms to generate many false candidate itemsets. In contrast, the performance of SWF will be less affected by the data skew since SWF employs the cumulative information for pruning false candidate itemsets in the early stage.

Extensive experiments are performed to assess the performance of SWF. As shown in the experimental results, SWF produces a significantly smaller amount of candidate 2-itemsets than FUP-based algorithms. In fact, the number of the candidate itemsets C_ks generated by SWF approaches to its theoretical minimum, i.e., the number of frequent k-itemsets, as the value of the minimal support increases. It is shown by our experiments that SWF in general significantly outperforms FUP-based algorithms. Explicitly, the execution time of SWF is, in orders of magnitude, smaller than those required by FUP-based algorithms. Sensitivity analysis on various parameters of the database is also conducted to provide many insights into algorithm SWF. The advantage of SWF over FUP-based algorithms becomes even more prominent not only as the amount of increased dataset increases but also as the size of the database increases. This is indeed an important feature for SWF to be practically used for the mining of an ongoing transaction database.

The rest of this paper is organized as follows. Preliminaries and related works are given in Section 2. Algorithm SWF is described in Section 3 with its correctness proved. Performance studies on various schemes are conducted in Section 4. This paper concludes with Section 5.

2. Preliminaries and related works

Let $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$ be a set of literals, called items. Let D be a set of transactions, where each transaction T is a set of items such that $T \subseteq \mathcal{I}$. Note that the quantities of items bought in a transaction are not considered, meaning that each item is a binary variable representing if an item was bought. Each transaction is associated with an identifier, called TID. Let X be a set of items. A transaction T is said to contain X if and only if $X \subseteq T$. An association rule is an implication of the form $X \Rightarrow Y$, where $X \subset \mathcal{I}$, $Y \subset \mathcal{I}$ and $X \cap Y = \phi$. The rule $X \Rightarrow Y$ holds in the transaction set D with *confidence* c if $c\%$ of transactions in D that contain X also contain Y. The rule $X \Rightarrow Y$ has *support* s in the transaction set D if $s\%$ of transactions in D contain $X \cup Y$. For a given pair of confidence and support thresholds, the problem of mining association rules is to find out all the association rules that have confidence and support greater than the corresponding thresholds. This problem can be reduced to the problem of finding all frequent itemsets for the same support threshold [1]. Before the description of algorithm SWF in Section 3, some related works are reviewed below.

2.1. Apriori-like algorithms

Most of the previous studies, including those is [1,17,31,32,35–37], belong to Apriori-like approaches. Basically, an Apriori-like approach is based on an anti-monotone Apriori heuristic [1], i.e., if any itemset of length k is not frequent in the database, its length $(k+1)$ super-itemset will never be frequent. The essential idea is to iteratively generate the set of candidate itemsets of length $(k+1)$ from the set of frequent itemsets of length k (for $k \geqslant 1$), and to check their corresponding occurrence frequencies in the database. As a result, if the largest *frequent* itemset is a j-itemset, then an Apriori-like algorithm may need to scan the database up to $(j+1)$ times.

In Apriori-like algorithms, C_3 is generated from $L_2 \bigstar L_2$, where C_3 is the set of all candidate itemsets of length 3 and L_2 is the set of all frequent itemset of length 2. In fact, a C_2 can be used to generate the candidate 3-itemsets. This technique is referred to as scan reduction in [4]. Clearly, a C_3' generated from $C_2 \bigstar C_2$, instead of from $L_2 \bigstar L_2$, will have a size greater than $|C_3|$ where C_3 is generated from $L_2 \bigstar L_2$. However, if $|C_3'|$ is not much larger than $|C_3|$, and both C_2 and C_3 can be stored in main memory, we can find L_2 and L_3 together when the next scan of the database is performed, thereby saving one round of database scan. It can be seen that using this concept, one can determine all L_ks by as few as two scans of the database (i.e., one initial scan to determine L_1 and a final scan to determine all other frequent itemsets), assuming that C'_k for $k \geqslant 3$ is generated from C'_{k-1} and all C'_k for $k > 2$ can be kept in the memory. In [5], the technique of scan-reduction was utilized and shown to result in prominent performance improvement.

2.2. Partition-based algorithms

The works in [18,20,38] are essentially based on a partition-based heuristic, i.e., *if X is a frequent itemset in database D which is divided into n partitions $p_1, p_2, \ldots, p_n$, then X must be a frequent itemset in at least one of the n partitions*. The partition algorithm in [18] divides $\mathcal{D}$ into n partitions, and processes one partition in main memory at a time. The algorithm first scans partition p_i, for $i = 1$ to n, to find the set of all local frequent itemsets in p_i, denoted as L^{p_i}. Then, by taking the union of L^{p_i} for $i = 1$ to n, a set of candidate itemsets over $\mathcal{D}$ is constructed, denoted as C^G. Based on the above partition-based heuristic, C^G is a superset of the set of all frequent itemsets in $\mathcal{D}$. Finally, the algorithm scans each partition for the second time to calculate the support of each itemset in C^G and to find out which candidate itemsets are really frequent itemsets in $\mathcal{D}$. Instead of constructing C^G by taking the union of L^{p_i}, for $i = 1$ to n, at the end of the first scan, some variations of the above partition algorithm are proposed in [20,38]. In [38], algorithm SPINC constructs C^G incrementally by adding L^{p_i} to C^G whenever L^{p_i} is available. SPINC starts the counting of occurrences for each candidate itemset $c \in C^G$ as soon as c is added to C^G. In [20], algorithm AS-CPA employs prior knowledge collected during the mining process to further reduce the number of candidate itemsets and to overcome the problem of data skew. However, these works were not devised to handle incremental updating of association rule.

2.3. FUP-based algorithms

Since it is costly to find the association rules in large databases, incremental updating techniques are desirable in order to avoid redoing data mining on the whole updated database. Basically, similar to that of Apriori, the framework of FUP, which can update the association rules in a database when new transactions are added to the database, contains a number of iterations [31,32]. The candidate sets at each iteration are generated based on the frequent itemsets found in the previous iteration. The key steps of FUP are listed below, where $\triangle^+$ denotes the added portion of an ongoing transaction database. (1) At each iteration, the supports of the size-k frequent itemsets in L are updated against the increment $\triangle^+$ to filter out those that are no longer in the updated database. (2) While scanning the increment, a set of candidate sets, C_k, is extracted from the transactions in $\triangle^+$, together with their supports in $\triangle^+$ counted. The supports of these sets in C_k are then updated against the original database to find the "new" frequent itemsets. (3) Many sets in C_k can be pruned away by checking their supports in $\triangle^+$ before the update against the original database starts. (4) The size of the updated database is reduced at each iteration by pruning away a few items from some transactions in the updated database.

The major idea is to reuse the information of the old frequent itemsets and to integrate the support information of the new frequent itemsets in order to substantially reduce the pool of candidate sets to be re-examined. An extension to FUP was reported in [32] and is referred to as FUP_2. In essence, FUP_2 is equivalent to FUP for the case of insertion, and is, however, a complementary algorithm of FUP for the case of deletion. Another FUP-based algorithm, call $\mathrm{FUP}_2\mathcal{H}$, was also devised in [32] to utilize the hash technique for performance improvement. As pointed out earlier, the existing FUP-based algorithms in general suffer from two inherent problems, namely (1) the occurrence of a potentially huge set of candidate itemsets, which is particularly critical for incremental mining since the candidate sets for the original database and the incremental portion are generated separately, and (2) the need of multiple scans of database.

3. SWF: incremental mining with sliding-window filtering

In essence, by partitioning a transaction database into several partitions, algorithm SWF employs a filtering threshold in each partition to deal with the candidate itemset generation. As described earlier, under SWF, the cumulative information in the prior phases is selectively carried over toward the generation of candidate itemsets in the subsequent phases. In the processing of a partition, a progressive candidate set of itemsets is generated by SWF. Explicitly, a progressive candidate set of itemsets is composed of the following two types of candidate itemsets, i.e., (1) the candidate itemsets that were carried over from the previous progressive candidate set in the previous phase and remain as candidate itemsets after the current partition is taken into consideration (such candidate itemsets are called type α candidate itemsets); and (2) the candidate itemsets that were not in the progressive candidate set in the previous phase but are newly selected after only taking the current data partition into account (such candidate itemsets are called type β candidate itemsets). As such, after the processing of a phase, algorithm SWF outputs a *cumulative filter*, denoted by *CF*, which consists of a progressive candidate set of itemsets, their occurrence counts and the corresponding partial support required. With these design considerations, algorithm SWF is shown to have very good performance for incremental mining. In Section 3.1, an illustrative example of SWF is presented. A detailed description of algorithm SWF is given in Section 3.2. The correctness of SWF is proved in Section 3.3.

3.1. An example of incremental mining by SWF

Algorithm SWF proposed can be best understood by the illustrative transaction database in Figs. 2 and 3 where a scenario of generating frequent itemsets from a transaction database for the incremental mining is given. The minimum transaction support is assumed to be $s = 40\%$. Without loss of generality, the incremental

$db^{1,3}$	$\triangle^-$	P_1	t_1	A	B	C				
			t_2	A					F	
			t_3	A	B	C		E		
	D^-	P_2	t_4	A	B		D	E		$db^{2,4}$
			t_5			C			F	
			t_6	A	B	C	D			
		P_3	t_7		B	C		E		
			t_8	A		C			F	
			t_9		B		D	E		
	$\triangle^+$	P_4	t_{10}		B		D	E	F	
			t_{11}				D	E	F	
			t_{12}	A		C				

Fig. 2. An illustrative transaction database.

mining problem can be decomposed into two procedures:

1. *Preprocessing procedure*: This procedure deals with mining on the original transaction database.
2. *Incremental procedure*: The procedure deals with the update of the frequent itemsets for an ongoing time-variant transaction database.

The preprocessing procedure is only utilized for the initial mining of association rules in the original database, e.g., $db^{1,n}$. For the generation of mining association rules in $db^{2,n+1}$, $db^{3,n+2}$, $db^{i,j}$, and so on, the incremental procedure is employed. Consider the database in Fig. 2. Assume that the original transaction database $db^{1,3}$ is segmented into three partitions, i.e., $\{P_1, P_2, P_3\}$, in the preprocessing procedure. Each partition is scanned sequentially for the generation of candidate 2-itemsets in the first scan of the database $db^{1,3}$. After scanning the first segment of three transactions, i.e., partition P_1, 2-itemsets {AB, AC, AE, AF, BC, BE, CE} are generated as shown in Fig. 3. In addition, each potential candidate itemset $c \in C_2$ has two attributes: (1) *c.start* which contains the identity of the starting partition when c was added to C_2, and (2) *c.count* which contains the number of occurrences of c since c was added to C_2. Since there are three transactions in P_1, the partial minimal support is $\lceil 3 \times 0.4 \rceil = 2$. Such a partial minimal support is called the *filtering threshold* in this paper. Itemsets whose occurrence

P_1			
	C_2	start	count
○	AB	1	2
○	AC	1	2
	AE	1	1
	AF	1	1
○	BC	1	2
	BE	1	1
	CE	1	1

P_2			
	C_2	start	count
○	AB	1	4
○	AC	1	3
○	AD	2	2
○	BC	1	3
○	BD	2	2
	BE	2	1
	CD	2	1
	CF	2	1
	DE	2	1

P_3			
	C_2	start	count
○	AB	1	4
○	AC	1	4
	AD	2	2
	AF	3	1
○	BC	1	4
○	BD	2	3
○	BE	3	2
	CE	3	1
	CF	3	1
	DE	3	1

Candidates in $db^{1,3}$:
{A}, {B}, {C}, {D}, {E}, {F}, {AB}, {AC}, {BC}, {BD}, {BE}, {ABC}
Large Itemsets in $db^{1,3}$:
{A}, {B}, {C}, {D}, {E}, {F}, {AB}, {AC}, {BC}, {BE}

$db^{1,3} - \triangle^{-} = D^{-}$			
	C_2	start	count
	AB	2	2
	AC	2	2
	BC	2	2
○	BD	2	3
○	BE	3	2

$D^{-} + \triangle^{+} = db^{2,4}$			
	C_2	start	count
	AC	4	1
○	BD	2	4
○	BE	3	3
	BF	4	1
○	DE	4	2
○	DF	4	2
○	EF	4	2

Candidates in $db^{1,3}$:
{A}, {B}, {C}, {D}, {E}, {F}, {BD}, {BE}, {DE}, {DF}, {EF}, {BDE}, {DEF}
Large Itemsets in $db^{2,4}$:
{A}, {B}, {C}, {D}, {E}, {F}, {BD}, {BE}, {DE}

Fig. 3. Large itemsets generation for the incremental mining with SWF.

counts are below the filtering threshold are removed. Then, as shown in Fig. 3, only {AB, AC, BC}, marked by "○", remain as candidate itemsets (of type β in this phase since they are newly generated) whose information is then carried over to the next phase of processing.

Similarly, after scanning partition P_2, the occurrence counts of potential candidate 2-itemsets are recorded (of type α and type β). From Fig. 3, it is noted that since there are also three transactions in P_2, the filtering threshold of those itemsets carried out from the previous phase (that become type α candidate itemsets in this phase) is $\lceil (3 + 3 \times 0.4 \rceil = 3$ and that of newly identified candidate itemsets (i.e., type β candidate itemsets) is $\lceil 3 \times 0.4 \rceil = 2$. It can be seen from Fig. 3 that we have five candidate itemsets in C_2 after the processing of partition P_2, and three of them are type α and two of them are type β.

Finally, partition P_3 is processed by algorithm SWF. The resulting candidate 2-itemsets are $C_2 = \{AB, AC, BC, BD, BE\}$ as shown in Fig. 3. Note that though appearing in the previous phase P_2, itemset {AD} is removed from C_2 once P_3 is taken into account since its occurrence count does not meet the filtering threshold then, i.e., $2 < 3$. However, we do have one new itemset, i.e., BE, which joins the C_2 as a type β candidate itemset. Consequently, we have five candidate 2-itemsets generated by SWF, and four of them are of type α

and one of them is of type β. Note that instead of 15 candidate itemsets that would be generated if Apriori were used,[1] only five candidate 2-itemsets are generated by SWF. The correctness of algorithm SWF will be formally proved later.

After generating C_2 from the first scan of database $db^{1,3}$, we employ the scan reduction technique and use C_2 to generate C_k ($k = 2, 3, \ldots, n$), where C_n is the candidate *last*-itemsets. It can be verified that a C_2 generated by SWF can be used to generate the candidate 3-itemsets and its sequential C'_{k-1} can be utilized to generate C'_k. Clearly, a C'_3 generated from $C_2 \bigstar C_2$, instead of from $L_2 \bigstar L_2$, will have a size greater than $|C_3|$ where C_3 is generated from $L_2 \bigstar L_2$. However, since the $|C_2|$ generated by SWF is very close to the theoretical minimum, i.e., $|L_2|$, the $|C'_3|$ is not much larger than $|C_3|$. Similarly, the $|C'_k|$ is close to $|C_k|$. All C'_k can be stored in main memory, and we can find L_k ($k = 1, 2, \ldots, n$) together when the second scan of the database $db^{1,3}$ is performed. Thus, only two scans of the original database $db^{1,3}$ are required in the preprocessing step. In addition, instead of recording all L_ks in main memory, we only have to keep C_2 in main memory for the subsequent incremental mining of an ongoing time variant transaction database.

The merit of SWF mainly lies in its incremental procedure. As depicted in Fig. 3, the mining database will be moved from $db^{1,3}$ to $db^{2,4}$. Thus, some transactions, i.e., t_1, t_2, and t_3, are *deleted* from the mining database and other transactions, i.e., t_{10}, t_{11}, and t_{12}, are *added*. For ease of exposition, this incremental step can also be divided into three sub-steps: (1) generating C_2 in $D^- = db^{1,3} - \triangle^-$, (2) generating C_2 in $db^{2,4} = D^- + \triangle^+$ and (3) scanning the database $db^{2,4}$ only once for the generation of all frequent itemsets L_k. In the first sub-step, $db^{1,3} - \triangle^- = D^-$, we check out the pruned partition P_1, and reduce the value of c.count and set $c.start = 2$ for those candidate itemsets c where $c.start = 1$. It can be seen that itemsets $\{AB, AC, BC\}$ were removed. Next, in the second sub-step, we scan the incremental transactions in P_4. The process in $D^- + \triangle^+ = db^{2,4}$ is similar to the operation of scanning partitions, e.g., P_2, in the preprocessing step. Three new itemsets, i.e., DE, DF, EF, join the C_2 after the scan of P_4 as type β candidate itemsets. Finally, in the third sub-step, we use C_2 to generate C'_k as mentioned above. With scanning $db^{2,4}$ only once, SWF obtains frequent itemsets $\{A, B, C, D, E, F, BD, BE, DE\}$ in $db^{2,4}$. As will be shown by experimental results later, the improvement achieved by algorithm SWF is even more prominent as the amount of the incremental portion increases and also as the size of the database $db^{i,j}$ increases.

[1] The details of the execution procedure by Apriori are omitted here. Interested readers are referred to [11].

3.2. *Algorithm of SWF*

For ease exposition, the meanings of various symbols used are given in Table 1. The preprocessing procedure and the incremental procedure of algorithm SWF are described in Sections 3.2.1 and 3.2.2, respectively.

3.2.1. *Preprocessing procedure of SWF*

The preprocessing procedure of Algorithm SWF is outlined below. Initially, the database $db^{1,n}$ is partitioned into n partitions by executing the preprocessing procedure (in Step 2), and *CF*, i.e., cumulative filter, is empty (in Step 3). Let $C_2^{i,j}$ be the set of progressive candidate 2-itemsets generated by database $db^{i,j}$. It is noted that instead

Table 1
Meanings of symbols used

$db^{i,j}$	Partition_database ($\mathcal{D}$) formed by a continuous region from partition P_i to partition P_j
s	Minimum support required
$\|P_k\|$	Number of transactions in partition P_k
$N_{p_k}(I)$	Number of transactions in partition P_k that contain itemset I
$\|db^{1,n}(I)\|$	Number of transactions in $db^{1,n}$ that contain itemset I
$C^{i,j}$	The set of progressive candidate itemsets generated by database $db^{i,j}$
$\triangle^-$	The deleted portion of an ongoing transaction database
D^-	The unchanged portion of an ongoing transaction database
$\triangle^+$	The added portion of an ongoing transaction database

of keeping $L_k s$ in the main memory, algorithm SWF only records $C_2^{1,n}$ which is generated by the preprocessing procedure to be used by the incremental procedure.

Preprocessing procedure of *Algorithm SWF*

```
1.  $n$ = Number of partitions;
2.  $|db^{1,n}| = \sum_{k=1,n} |P_k|$;
3.  $CF = \emptyset$;
4.  begin for $k = 1$ to $n$        // 1st scan of $db^{1,n}$
5.     begin for each 2-itemset $I \in P_k$
6.        if $(I \notin CF)$
7.           $I.count = N_{p_k}(I)$;
8.           $I.start = k$;
9.           if $(I.count \geqslant s*|P_k|)$
10.             $CF = CF \cup I$;
11.       if $(I \in CF)$
12.          $I.count = I.count + N_{p_k}(I)$;
13.          if $(I.count < \lceil s*\sum_{m=I.start,k} |P_m| \rceil)$
14.             $CF = CF - I$;
15.    end
16. end
17. select $C_2^{1,n}$ from $I$ where $I \in CF$;
18. keep $C_2^{1,n}$ in main memory;
19. $h = 2$;      //$C_1$ is given
20. begin while $(C_h^{1,n} \neq \emptyset)$      //Database scan reduction
21.    $C_{h+1}^{1,n} = C_h^{1,n} \bigstar C_h^{1,n}$;
22.    $h = h + 1$;
23. end
24. refresh $I.count = 0$ where $I \in C^{1,n}$;      //where $C^{1,n} = \bigcup_h C_h^{1,n}$
25. begin for $k = 1$ to $n$       //2nd scan of $db^{1,n}$
26.    for each itemset $I \in C^{1,n}$
27.       $I.count = I.count + N_{p_k}(I)$;
28. end
29. for each itemset $I \in C^{1,n}$
30.    if $(I.count \geqslant \lceil s*|db^{1,n}| \rceil)$
31.       $L = L \cup I$;
32. end
33. return $L$;
```

From Step 4 to Step 16, the algorithm processes one partition at a time for all partitions. When partition P_i is processed, each potential candidate 2-itemset is read and saved to *CF*. The number of occurrences of an itemset I and its starting partition are recorded in *I.count* and *I.start*, respectively. An itemset, whose $I.count \geqslant \lceil s*\sum_{m=I.start,k} |P_m| \rceil$, will be kept in *CF*. Next, we select $C_2^{1,n}$ from I where $I \in CF$ and keep $C_2^{1,n}$ in main memory for the subsequent incremental procedure. With employing the scan reduction technique from Step 19 to Step 23, $C_h^{1,n}$s ($h \geqslant 3$) are generated in main memory. After refreshing $I.count = 0$ where $I \in C^{1,n}$, we begin the last scan of database for the preprocessing procedure from Step 25 to Step 28. Finally, those itemsets whose $I.count \geqslant \lceil s*|db^{1,n}| \rceil$ are the frequent itemsets.

3.2.2. Incremental procedure of SWF

As shown in Table 1, D^- indicates the unchanged portion of an ongoing transaction database. The deleted and added portions of an ongoing transaction database are denoted by $\triangle^-$ and $\triangle^+$, respectively. It is worth mentioning that the sizes of $\triangle^+$ and $\triangle^-$, i.e., $|\triangle^+|$ and $|\triangle^-|$, respectively, are not required to be the same. The incremental procedure of SWF is devised to maintain frequent itemsets efficiently and effectively. This procedure is outlined below.

Incremental procedure of *Algorithm SWF*

```
1.  Original database = $db^{m,n}$;
2.  New database = $db^{i,j}$;
3.  Database removed $\triangle^- = \sum_{k=m,i-1} P_k$;
4.  Database increased $\triangle^+ = \sum_{k=n+1,j} P_k$;
5.  $D^- = \sum_{k=i,n} P_k$;
6.  $db^{i,j} = db^{m,n} - \triangle^- + \triangle^+$;
7.  loading $C_2^{m,n}$ of $db^{m,n}$ into $CF$ where $I \in C_2^{m,n}$;
8.  begin for $k = m$ to $i - 1$       // one scan of $\triangle^-$
9.     begin for each 2-itemset $I \in P_k$
10.       if $(I \in CF$ and $I.start \leqslant k)$
11.          $I.count = I.count - N_{p_k}(I)$;
12.          $I.start = k + 1$;
13.          if $(I.count < \lceil s*\sum_{m=I.start,n} |P_m| \rceil)$
14.             $CF = CF - I$;
15.    end
16. end
17. begin for $k = n + 1$ to $j$       // one scan of $\triangle^+$
18.    begin for each 2-itemset $I \in P_k$
19.       if ( $I \notin CF$)
20.          $I.count = N_{p_k}(I)$;
21.          $I.start = k$;
22.          if $(I.count \geqslant s*|P_k|)$
23.             $CF = CF \cup I$;
```

```
24.         if ($I \in CF$)
25.             $I.count = I.count + N_{p_k}(I)$;
26.             if ($I.count < \lceil s * \sum_{m=I.start,k} |P_m| \rceil$)
27.                 $CF = CF - I$;
28.     end
29. end
30. select $C_2^{i,j}$ from $I$ where $I \in CF$;
31. keep $C_2^{i,j}$ in main memory
32. $h = 2$        //$C_1$ is well known.
33. begin while ($C_h^{i,j} \neq \emptyset$)        //Database scan reduction
34.     $C_{h+1}^{i,j} = C_h^{i,j} \bigstar C_h^{i,j}$;
35.     $h = h + 1$;
36. end
37. refresh $I.count = 0$ where $I \in C^{i,j}$;        //where $C^{1,n} = \bigcup_h C_h^{1,n}$
38. begin for $k = i$ to $j$        //only one scan of $db^{i,j}$
39.     for each itemset $I \in C^{i,j}$
40.         $I.count = I.count + N_{p_k}(I)$;
41. end
42. for each itemset $I \in C^{i,j}$
43.     if ($I.count \geqslant \lceil s * |db^{i,j}| \rceil$)
44.         $L = L \cup I$;
45. end
46. return $L$;
```

As mentioned before, this incremental step can also be divided into three sub-steps: (1) generating C_2 in D^-, (2) generating C_2 in $D^- + \triangle^+$ and (3) scanning the database $D^- + \triangle^+$ only once for the generation of all frequent itemsets L_k. Initially, after some update activities, old transactions $\triangle^-$ are removed from the database $db^{m,n}$ and new transactions $\triangle^+$ are added (in Step 6). Note that $\triangle^- \subset db^{m,n}$. Denote the updated database as $db^{i,j}$. Note that $db^{i,j} = db^{m,n} - \triangle^- + \triangle^+$. We denote the unchanged transactions by $D^- = db^{m,n} - \triangle^- = db^{i,j} - \triangle^+$. After loading $C_2^{m,n}$ of $db^{m,n}$ into CF where $I \in C_2^{m,n}$, we start the first sub-step, i.e., generating C_2 in $D^- = db^{m,n} - \triangle^-$. This sub-step tries to reverse the cumulative processing which is described in the preprocessing procedure. From Step 8 to Step 16, we prune the occurrences of an itemset I, which appeared before partition P_i, by deleting the value *I.count* where $I \in CF$ and $I.start < i$. Next, from Step 17 to Step 36, similarly to the cumulative processing in Section 3.2.1, the second sub-step generates new potential $C_2^{i,j}$ in $db^{i,j} = D^- + \triangle^+$ and employs the scan reduction technique to generate $C_h^{i,j}$s from $C_2^{i,j}$. Finally, to generate new L_ks in the updated database, we scan $db^{i,j}$ for only once in the incremental procedure to maintain frequent itemsets. Note that $C_2^{i,j}$ is kept in main memory for the next generation of incremental mining. Noted that, it is easy to extend algorithm SWF, so that after generating $C_2^{i,j}$, it takes one more database scan to obtain $L_2^{i,j}$ and then generate $C^{i,j}$ directly from $L_2^{i,j}$. Similarly, we can generate $C^{i,j}$ from $L_k^{i,j}$ by $(k-1)$ more database scans. This extension is especially useful for some extremely distributed data set, in which, L_k, where $k > 2$, is much larger then L_2.

Note that SWF is able to filter out false candidate itemsets in P_i with a hash table. Same as in [17], using a hash table to prune candidate 2-itemsets, i.e., C_2, in each accumulative ongoing partition set P_i of transaction database, the CPU and memory overhead of SWF can be further reduced. As will be validated by our experimental studies, SWF indeed provides an efficient solution for incremental mining, which is, in our opinion, important for mining the record-based databases whose data are being frequently and continuously added, such as Web log records, stock market data, grocery sales data, and transactions in electronic commerce, to name a few.

3.3. Correctness of SWF

With the above two procedures described, we now examine the correctness and effectiveness of algorithm SWF. Let $N_{p_k}(I)$ be the number of transactions in partition P_k that contain itemset I, and $|P_k|$ is the number of transactions in partition P_k. Also, let $db^{i,j}$ denote the part of the transaction database formed by a continuous region from partition P_i to partition P_j, and $|db^{i,j}| = \sum_{k=i,j} |P_k|$. We can then define the region ratio of an itemset as follows.

Definition. A region ratio of an itemset I for the transaction database $db^{i,j}$, denoted by $r_{i,j}(I)$, is $r_{i,j}(I) = (\sum_{k=i,j} N_{p_k}(I))/|db^{i,j}|$.

In essence, the region ratio of an itemset is the support of that itemset if only the part of transaction database $db^{i,j}$ is considered.

Lemma 1. *An itemset I remains in the CF after the processing of partition P_j if and only if there exists an i such that for any integer k in the interval $[i,j]$, $r_{i,k}(I) \geqslant s$, where s is the minimal support required.*

Proof. We shall prove the "if" condition first. Consider the following two cases. First, suppose the itemset I is not in the progressive candidate set *before* the processing of partition P_i. Since $r_{i,i}(I) \geqslant s$, itemset I will be selected as a type β candidate itemset by SWF after the processing of partition P_i. On the other hand, if the itemset I is already in the progressive candidate set before the processing of partition P_i, itemset I will remain as a type α candidate itemset by SWF. Clearly, for the above two cases, itemset I will remain in *CF* throughout the processing from P_i to P_j since for any integer k in the interval $[i,j]$, $r_{i,k}(I) \geqslant s$.

We now prove the "only if" condition, i.e., if I remains in *CF* after the processing of partition P_j then there exists an i such that for any k in the interval $[i,j]$, $r_{i,k}(I) \geqslant s$. Note that itemset I can be either type α or type β candidate itemset in the *CF* after the processing of partition P_j. Suppose I is a type β candidate itemset there, then this implication follows by setting $j = i$ since $r_{i,i}(I) \geqslant s$. On the other hand, suppose that I is a type α candidate itemset after the processing of P_j, which means itemset I has become a type β candidate itemset in a previous phase. Then, we shall trace backward the type of itemset I from partition P_j (i.e., looking over P_j, P_{j-1}, P_{j-2} and so forth) until the partition that records itemset I as a type β candidate itemset is *first* encountered. (It should be noted that there could be two *discontinuous* regions that record itemset I in the *CF*, which means that an itemset may get on and off the progressive candidate set through the processing of partitions. This in turn means that an itemset may appear as a type β candidate itemset more than once. Such a scenario occurs for the itemset BE in the example in Fig. 3.) By referring the partition identified above as partition P_i, we have, for any k in the interval $[i,j]$, $r_{i,k}(I) \geqslant s$, completing the proof of this lemma. □

Lemma 1 leads to Lemma 2 below.

Lemma 2. *An itemset I remains in CF after the processing of partition P_j if and only if there exists an i such that $r_{i,j}(I) \geqslant s$, where s is the minimal support required.*

Proof. It can be seen that the proof of "only if" condition follows directly from Lemma 1. We now prove the "if" condition of this lemma. If there exists an i such that $r_{i,j}(I) \geqslant s$ then we let t be the largest x such that $r_{i,x}(I) < s$. If such a t does not exist, it follows from Lemma 1 that itemset I will remain in *CF* after the processing of partition P_j. If such a t exists, we have $r_{t+1,j}(I) \geqslant s$ since $r_{i,t}(I) < s$ and $r_{i,j}(I) \geqslant s$. It again follows from Lemma 1 that itemset I will remain in *CF* after the processing of partition P_j. This lemma follows. □

Lemma 2 leads to the following theorem which states the correctness of algorithm SWF.

Theorem 1. *If an itemset I is a frequent itemset, then I will be in the progressive candidate set of itemsets produced by algorithm SWF.*

Proof. Let n be the number of partitions of the transaction database. Since the itemset I is a frequent itemset, we have $r_{1,n}(I) \geqslant s$, which is in essence a special case of Lemma 2 for $i = 1$ and $j = n$, proving this theorem. □

It follows from Theorem 1 that the frequent itemsets generated by SWF are the same as those produced by existing association rule mining algorithms such as Apriori. Furthermore, we let $C^{i,j}$, where $i \leqslant j$, be the set of progressive candidate itemsets generated by algorithm SWF with respect to database $db^{i,j}$ after the processing of P_j. We then have the following lemma.

Lemma 3. *For $i \leqslant k \leqslant j$, then $C^{k,j} \subset C^{i,j}$.*

Proof. Assume that there exists an itemset $I \in C^{k,j}$. From the "only if" implication of Lemma 2, it follows that there exists an h such that $r_{h,j}(I) \geqslant s$, where $k \leqslant h \leqslant j$. Since $i \leqslant k \leqslant j$, we have $i \leqslant h \leqslant j$. Then, according to the "if" implication of Lemma 2, itemset I is also in $C^{i,j}$, i.e., $I \in C^{i,j}$. The fact that $C^{k,j} \subset C^{i,j}$ follows. □

Lemma 3 leads to the following theorem which states the effectiveness of SWF for incremental mining.

Theorem 2. If an itemset I is a frequent itemset with respect to the database $db^{i+1,j+1}$, then itemset I is either in $C^{i,j}$ or will be a type β candidate itemset after the processing of partition P_{j+1}.

Proof. If an itemset I is a frequent itemset with respect to the database $db^{i+1,j+1}$, we then have $r_{i+1,j+1}(I) \geqslant s$. Three cases for $r_{i+1,j+1}(I) \geqslant s$ are considered. The first case is $r_{i+1,j}(I) \geqslant s$ and $r_{j+1,j+1}(I) \geqslant s$, and the second one is $r_{i+1,j}(I) \geqslant s$ and $r_{j+1,j+1}(I) \leqslant s$. From Theorem 1, it follows that in the above two cases, $I \in C^{i+1,j}$, which in turn implies that $I \in C^{i,j}$ since we have $C^{i+1,j} \subset C^{i,j}$ by Lemma 3.

Consider the third case where $r_{i+1,j}(I) < s$ and $r_{j+1,j+1}(I) \geqslant s$. If $r_{i+1,j}(I) < s$ and $I \notin C^{i+1,j}$, then itemset I will be a type β candidate itemset after the processing of partition P_{j+1} since $r_{j+1,j+1}(I) \geqslant s$. On the other hand, if $r_{i+1,j}(I) < s$ but $I \in C^{i+1,j}$, we also get $I \in C^{i,j}$, from Lemma 3. This theorem is thus proved. $\square$

Note that any itemset I that is a frequent itemset with respect to $db^{i+1,j+1}$ and has appeared in $C^{i,j}$ will be identified as a type α candidate itemset after the processing of partition P_{j+1}. From Theorem 2 and this fact, it follows that the cumulative filters of algorithm SWF can be determined in a progressive manner without missing any possible frequent itemsets even in the presence of the need of mining an ongoing time variant transaction database.

4. Experimental studies

To assess the performance of algorithm SWF, we performed several experiments on a computer with a CPU clock rate of 450 MHz and 512 MB of main memory. The transaction data resides in the NTFS file system and is stored on a 30GB IDE 3.5″ drive with a measured sequential throughput of 10 MB/s. The simulation program was coded in C++. The methods used to generate synthetic data are described in Section 4.1. The performance comparison of SWF, FUP_2 and Apriori is presented in Section 4.2. Section 4.3 shows the I/O cost among SWF, $FUP_2\mathcal{H}$ and Apriori. We conduct some experiments on examining CPU and memory overhead in Section 4.4. Results on scaleup experiments are presented in Section 4.5.

4.1. Generation of synthetic workload

For obtaining reliable experimental results, the method to generate synthetic transactions we employed in this study is similar to the ones used in prior works [17,31,33,34]. Explicitly, we generated several different transaction databases from a set of potentially frequent itemsets to evaluate the performance of SWF. These transactions mimic the transactions in the retailing environment. Note that the efficiency of algorithm SWF has been evaluated by some real databases, such as Web log records and grocery sales data. However, we show the experimental results from synthetic transaction data so that the work relevant to data cleaning, which is in fact application dependent and also orthogonal to the incremental technique proposed, is hence omitted for clarity. Further, more sensitivity analysis can then be conducted by using the synthetic transaction data. Each database consists of $|D|$ transactions, and on the average, each transaction has $|T|$ items. Table 2 summarizes the meanings of various parameters used in the experiments. The mean of the correlation level is set to 0.25 for our experiments.

Recall that the sizes of $|\triangle^+|$ and $|\triangle^-|$ are not required to be the same for the execution of SWF. Without loss of generality, we set $|d| = |\triangle^+| = |\triangle^-|$ for simplicity. Thus, by denoting the original database as $db^{1,n}$ and the new mining database as $db^{i,j}$, we have $|db^{i,j}| = |db^{1,n} - \triangle^- + \triangle^+| = |\mathcal{D}|$, where $\triangle^- = db^{1,i-1}$ and $\triangle^+ = db^{n+1,j}$. In the following, we use the notation $Tx - Iy - Dm - dn$ to represent a database in which $D = m$ thousands, $d = n$ thousands, $|T| = x$, and $|I| = y$. We compare relative performance of three methods, i.e., Apriori, FUP-based algorithms and SWF.

As mentioned before, without any provision for incremental mining, Apriori algorithm has to re-run the association rule mining algorithm on the

Table 2
Meanings of various parameters

$\vert\mathcal{D}\vert$	Number of transactions in the database
$\vert\triangle^+\vert$	Number of added transactions
$\vert\triangle^-\vert$	Number of deleted transactions
$\vert d\vert$	Number of incremental transactions
$\vert T\vert$	Average size of the transactions
$\vert I\vert$	Average size of the maximal potentially frequent itemsets
$\vert L\vert$	Number of maximal potentially frequent itemsets
N	Number of items

whole updated database. As reported in [31,32], with reducing the candidate itemsets, FUP-based algorithms outperform Apriori. As will be shown by our experimental results, with the sliding window technique that carries cumulative information selectively, the execution time of SWF is, in orders of magnitude, smaller than those required by FUP-based algorithms. In order to conduct our experiments on a database of size $db^{i,j}$ with an increment of $\triangle^+$ and a removal of $\triangle^-$, a database of $db^{1,j}$ is first generated and then $db^{1,i-1}$, $db^{1,n}$, $db^{n+1,j}$, and $db^{i,j}$ are produced separately.

4.2. Experiment one: relative performance

We first conducted several experiments to evaluate the relative performance of Apriori, FUP_2 and SWF. As shown in Fig. 4, the experimental results are consistent from one to another for various values of $|L|$ and N on dataset $T10-I4-D100-d10$. For interest of space, we only report the results on $|L|=2000$ and $N=10000$ in the following experiments. Fig. 5 shows the relative execution times for the three algorithms as the minimum support threshold is decreased from 1% support to 0.1% support. When the support threshold is high, there are only a limited number of frequent itemsets produced. However, as the support threshold decreases, the performance difference becomes prominent in that SWF significantly outperforms both FUP_2 and Apriori. As shown in Fig. 5, SWF leads to prominent performance improvement for various sizes of $|T|$, $|I|$ and $|d|$. Explicitly, SWF is in orders of magnitude faster than FUP_2, and the margin grows as the minimum support threshold decreases. Note that from our experimental results, the difference between FUP_2 and Apriori is consistent with that observed in [32]. In fact, SWF outperforms FUP_2 and Apriori in both CPU and I/O costs, which are evaluated next.

4.3. Experiment two: evaluation of I/O cost

To evaluate the corresponding of I/O cost, same as in [24], we assume that each sequential read of a byte of data consumes one unit of I/O cost and each random read of a byte of data consumes two units of I/O cost. Fig. 6 shows the number of database scans and the I/O costs of Apriori, $FUP_2\mathcal{H}$, i.e., hash-type FUP in [32], and SWF over data sets $T10-I4-D100-d10$ and $T10-I4-D200-d20$. As shown in Fig. 6, SWF outperforms Apriori and $FUP_2\mathcal{H}$ where without loss of generality a hash table of 250 000 entries is employed for those methods. Note that the large amount of database scans is the performance bottleneck when the database size does not fit into main memory. In view of that, SWF is advantageous since only one scan of the updated database is required, which is independent of the variance in minimum supports.

4.4. Experiment three: reduction of CPU and memory overhead

As explained before, SWF substantially reduces the number of candidate itemsets generated. The effect is particularly important for the candidate 2-itemsets. The experimental results in Fig. 7 show the candidate itemsets generated by Apriori, $FUP_2\mathcal{H}$, and SWF across the whole processing on the datasets $T10-I4-D100-d10$ and $T10-I4-D200-d20$ with minimum support threshold $s=0.1\%$. As shown in Fig. 7, SWF leads to a 99% candidate reduction rate in C_2 when being compared to Apriori, and leads to a 93% candidate reduction rate in C_2 when being compared to $FUP_2\mathcal{H}$. Similar phenomena were observed when other datasets were used. This feature of SWF enables it to efficiently reduce the CPU and memory overhead. Note that the number of candidate 2-itemsets produced by

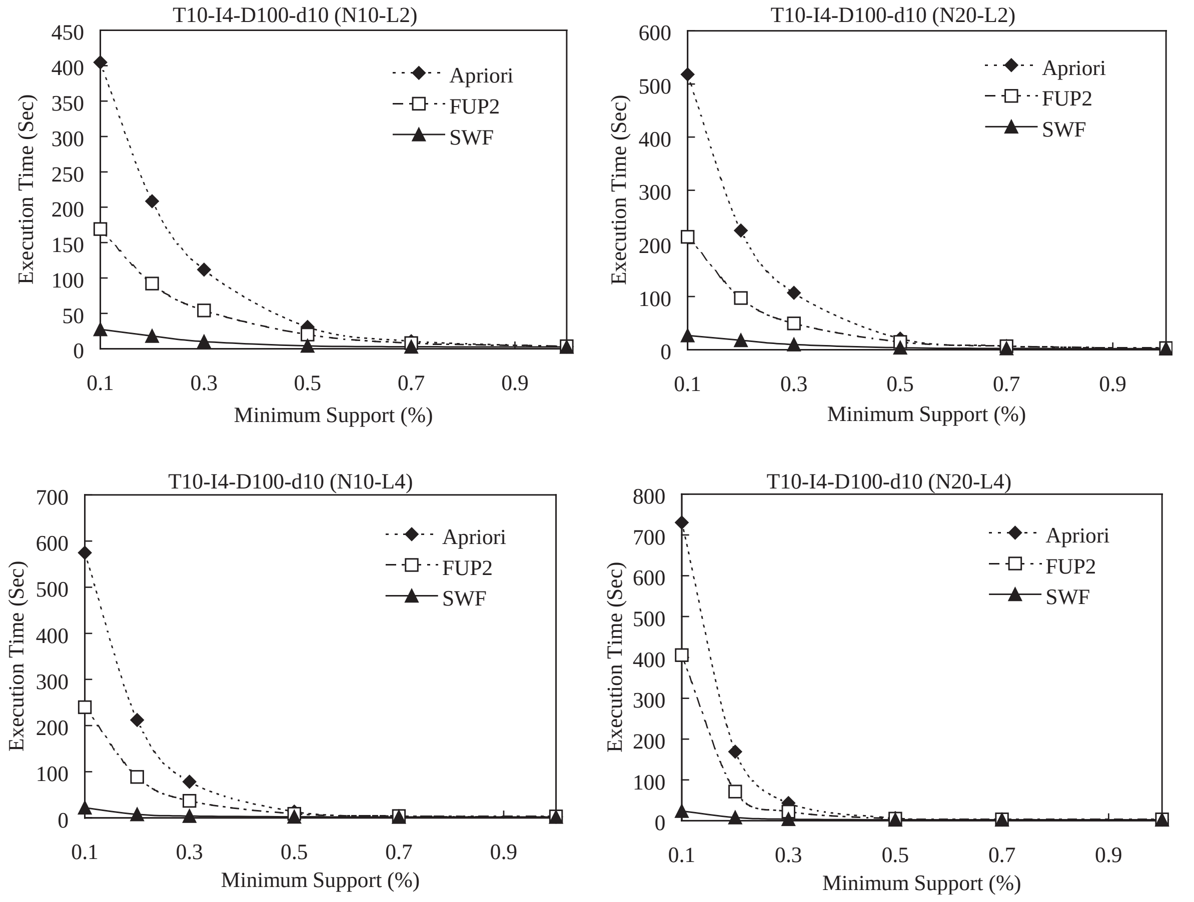


Fig. 4. Extensive analysis of various values of N and $|L|$.

SWF approaches to its theoretical minimum, i.e., the number of frequent 2-itemsets. Recall that the C_3 in either Apriori or $\mathrm{FUP}_2\mathcal{H}$ has to be obtained by L_2 due to the large size of their C_2. As shown in Fig. 7, the value of $|C_k|$ ($k \geqslant 3$) is only slightly larger than that of Apriori or $\mathrm{FUP}_2\mathcal{H}$, even though SWF only employs C_2 to generate C_ks, thus fully exploiting the benefit of scan reduction.

4.5. Experiment four: scaleup performance

In this experiment, we examine the scaleup performance of algorithm SWF. The scale-up results for different selected datasets are obtained. Fig. 8 shows the scaleup performance of algorithm SWF as the values of $|D|$ and $|d|$ increase. Three different minimum supports are considered. We obtained the results for the dataset $T10-I4-Dm-d10$ when the number of customers increases from 100,000 to one million. The execution times are normalized with respect to the times for the 100,000 transactions dataset in the Fig. 8a. The second scaleup experiment with the dataset $T10-I4-D1000-dn$ shows the performance results of SWF when the number of transactions in the increased dataset varies from 50 thousands to 300 thousands. The execution times are normalized with respect to the times for the 50,000 increased transaction dataset in the Fig. 8b. Note that, as shown in Fig. 8b the execution time only slightly increases with the growth of the incremental size, showing good scalability of SWF.

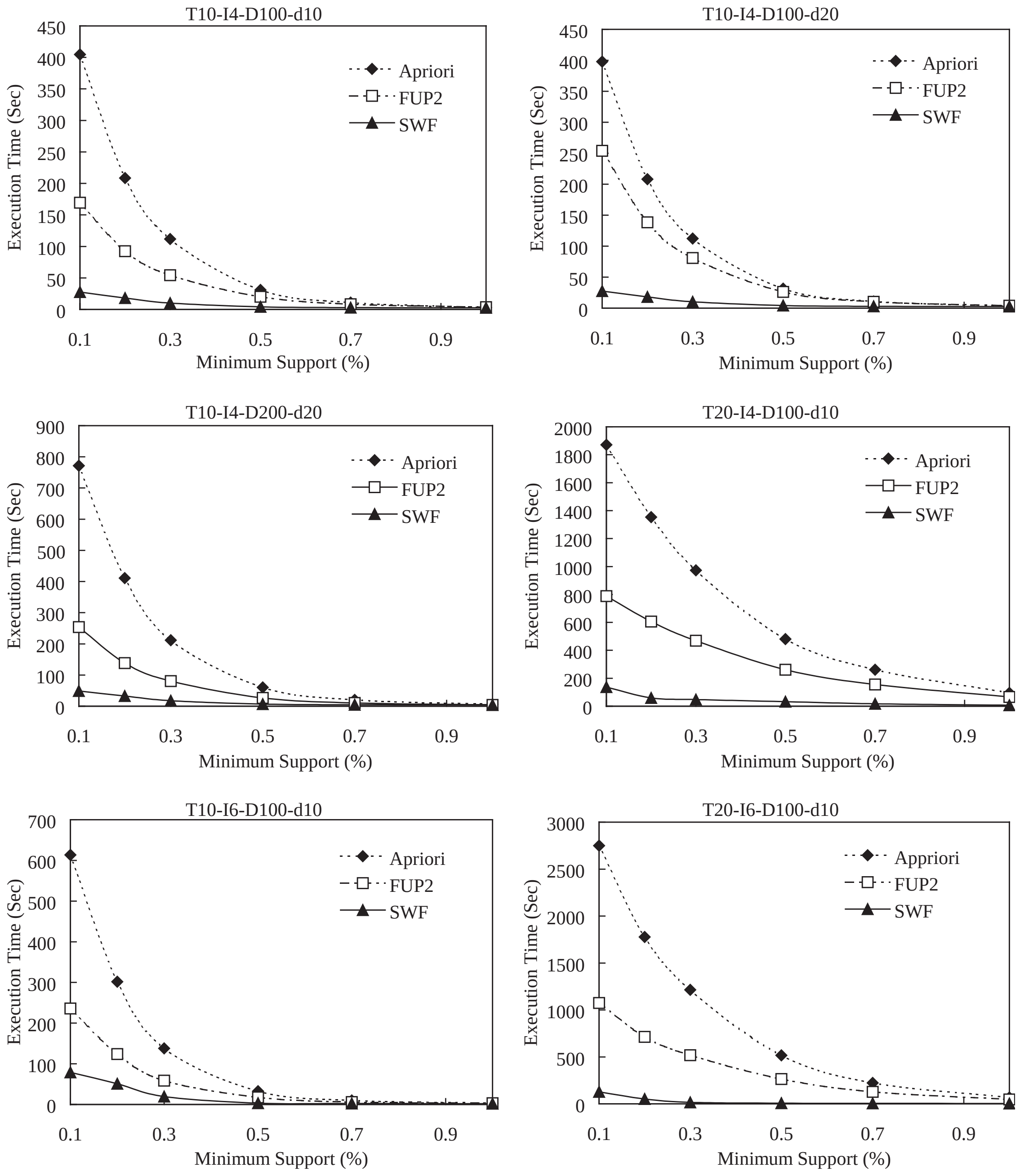
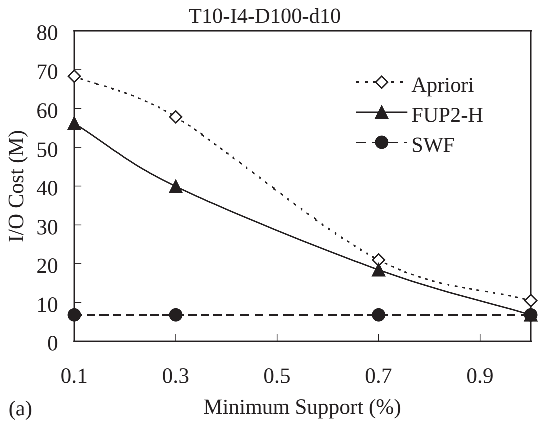


Fig. 5. Relative performance.

To further understand the impact of |*D*| and |*d*| to the relative performance of algorithms SWF and FUP-based algorithms, we conduct the scale-up experiments for both SWF and FUP$_2$ with two minimum support thresholds 0.2% and 0.4%. The results are shown in Fig. 9 where the value in *y*-axis corresponds to the ratio of the execution time of SWF to that of FUP$_2$. Fig. 9a shows the



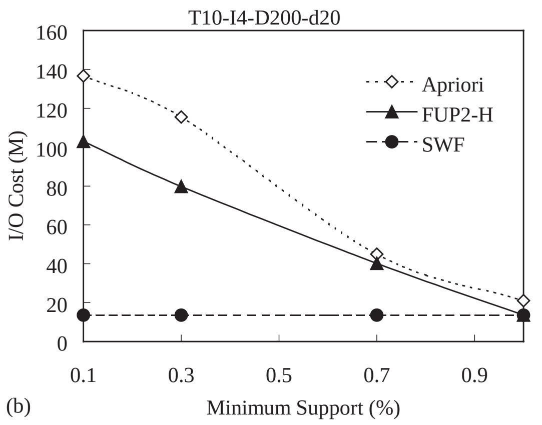


Fig. 6. I/O cost performance: (a) IO cost performance over data set T10-I4-D100-d10; (b) IO cost performance over data set T10-I4-D200-d20.

T10-I4-D100-d10				
Candidates	Apriori	FUP$_2$H	SWF	Freq. Itemsets
C_2	3399528	104145	7482	L_2=6656
C_3	8353	8353	9241	L_3=8135
C_4	7882	7882	8679	L_4=7616
C_5	6762	6382	7162	L_5=6077
C_6	5437	4709	5578	L_6=4658
C_7	3918	3417	3951	L_7=3412

(a)

T10-I4-D200-d20				
Candidates	Apriori	FUP$_2$H	SWF	Freq. Itemsets
C_2	3430890	105848	7632	L_2=6641
C_3	8332	8332	9468	L_3=8021
C_4	7752	7752	8996	L_4=7507
C_5	6406	6406	7510	L_5=5819
C_6	4643	4643	5852	L_6=4622
C_7	3421	3421	4100	L_7=3416

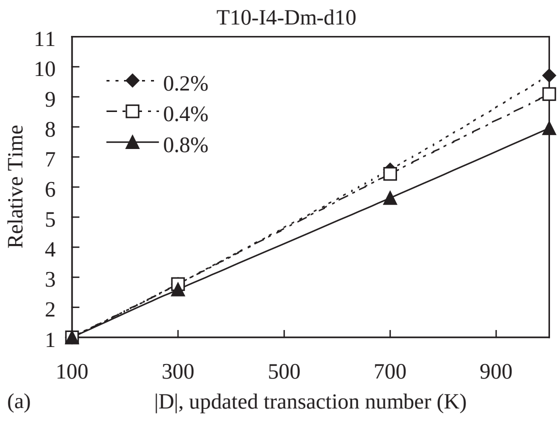
(b)

Fig. 7. Reduction on candidate itemsets when datasets (a) T10-I4-D100-d10 and (b) T10-I4-D200-d20 were used.

referenced ratio obtained from an updated database over datasets of $T10-I4-Dm-d(m/10)$. With the value $|D|/|d|=10$, the execution-time-ratio of SWF to FUP$_2$ decreases when the amount of updated database $|D|$ grows larger, meaning that the advantage of SWF over FUP$_2$ increases as the database size increases. Fig. 9b shows the execution-time-ratio for different values of $|d|$. It can be seen that since the size of $|d|$ has less influence on the performance of SWF, the execution-time-ratio becomes smaller with the growth of the incremental transaction number $|d|$. This also implies that the advantage of SWF over FUP$_2$ becomes even more prominent as the amount of incremental portion increases.

5. Conclusion

We explored in this paper an efficient sliding-window filtering algorithm for incremental mining of association rules. Under SWF, the cumulative information of mining previous partitions is selectively carried over toward the generation of candidate itemsets for the subsequent partitions. Algorithm SWF not only significantly reduces I/O and CPU cost by the concepts of cumulative filtering and scan reduction techniques but also effectively controls memory utilization by the technique of sliding-window partition. More importantly, SWF is particularly powerful for efficient incremental mining for an ongoing time-variant transaction database. The correctness of SWF is proved and some of its theoretical properties are derived. Extensive simulations have been performed to evaluate performance of algorithm SWF. Sensitivity analysis of various parameters was conducted to provide many insights into SWF. It was noted that the improvement achieved by SWF increases as the increased



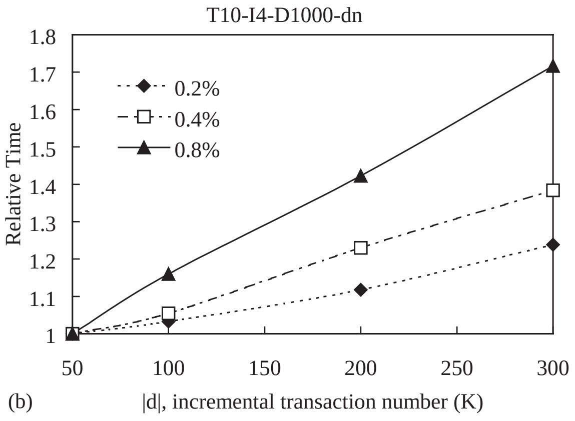
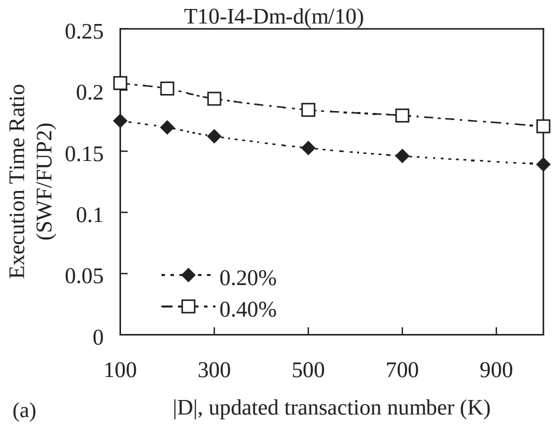


Fig. 8. Scaleup performance of SWF: (a) various value of $|D|$; (b) various value of $|d|$.



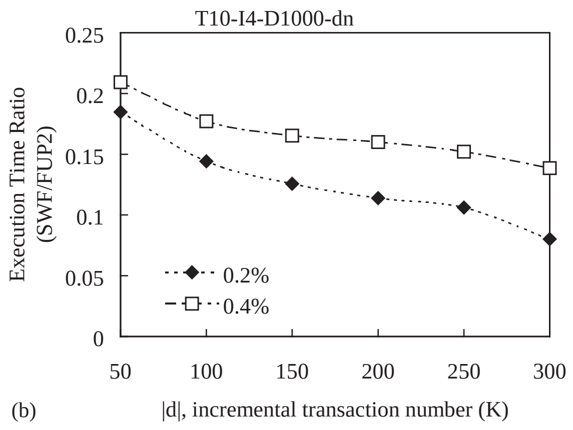


Fig. 9. Scaleup performance with the execution time ratio between SWF and FUP: (a) execution time ratio in various value of $|D|$: (b) execution time ratio in various value of $|d|$.

portion of the dataset increases and also as the size of the database increases.

References

[1] R. Agrawal, T. Imielinski, A. Swami, Mining association rules between sets of items in large databases, Proceedings of the ACM SIGMOD, May 1993, pp. 207–216.

[2] R. Agrawal, R. Srikant, Mining sequential patterns, Proceedings of the 11th International Conference on Data Engineering, March 1995, pp. 3–14.

[3] J.M. Ale, G. Rossi, An approach to discovering temporal association rules, ACM Symposium on Applied Computing, 2000.

[4] M.-S. Chen, J. Han, P.S. Yu, Data mining: an overview from database perspective, IEEE Trans. Knowledge Data Eng. 8 (6) (1996) 866–883.

[5] M.-S. Chen, J.-S. Park, P.S. Yu, Efficient data mining for path traversal patterns, IEEE Trans. Knowledge Data Eng. 10 (2) (1998) 209–221.

[6] X. Chen, I. Petr, Discovering temporal association rules: algorithms, language and system, Proceedings of 2000 International Conference on Data Engineering, 2000.

[7] J. Han, G. Dong, Y. Yin, Efficient mining of partial periodic patterns in time series database, in: Proceedings of the 15th International Conference on Data Engineering, March 1999, pp. 106–115.

[8] R.T. Ng, J. Han, Efficient and effective clustering methods for spatial data mining, Proceedings of the 18th Interna-

tional Conference on Very Large Data Bases, September 1994, pp. 144–155.

[9] K. Wang, S.Q. Zhou, S.C. Liew, Building hierarchical classifiers using class proximity, Proceedings of 1999 International Conference on Very Large Data Bases, 1999, pp. 363–374.

[10] C. Yang, U. Fayyad, P. Bradley, Efficient discovery of error-tolerant frequent itemsets in high dimensions, The Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2001.

[11] R. Agrawal, R. Srikant, Fast algorithms for mining association rules in large databases, Proceedings of the 20th International Conference on Very Large Data Bases, September 1994, pp. 478–499.

[12] E. Cohen, et al., Finding interesting associations without support pruning, IEEE Trans. Knowledge Data Eng. (2001) 64–78.

[13] J. Han, Y. Fu, Discovery of multiple-level association rules from large databases, Proceedings of the 21st International Conference on Very Large Data Bases, September 1995, pp. 420–431.

[14] J. Han, J. Pei, Y. Yin, Mining frequent patterns without candidate generation, Proceedings of 2000 ACM-SIGMOD International Conference on Management of Data, May 2000, pp. 486–493.

[15] B. Liu, W. Hsu, Y. Ma, Mining association rules with multiple minimum supports, Proceedings of 1999 International Conference on Knowledge Discovery and Data Mining, August 1999.

[16] H. Mannila, H. Toivonen, A. Inkeri Verkamo, Efficient algorithms for discovering association rules, Proceedings of AAAI Workshop on Knowledge Discovery in Databases, July, 1994, pp. 181–192.

[17] J.-S. Park, M.-S. Chen, P.S. Yu, Using a hash-based method with transaction trimming for mining association rules, IEEE Trans. Knowledge and Data Eng. 9 (5) (1997) 813–825.

[18] A. Savasere, E. Omiecinski, S. Navathe, An efficient algorithm for mining association rules in large databases, Proceedings of the 21st International Conference on Very Large Data Bases, September 1995, pp. 432–444.

[19] K. Wang, Y. He, J. Han, Mining frequent itemsets using support constraints, Proceedings of 2000 International Conference on Very Large Data Bases, September 2000.

[20] J.-L. Lin, M.H. Dunham, Mining association rules: anti-skew algorithms, Proceedings of 1998 International Conference on Data Engineering, 1998, pp. 486–493.

[21] R. Agarwal, C. Aggarwal, V.V.V. Prasad, A tree projection algorithm for generation of frequent itemsets, J. Parallel Distributed Comput. (special issue on High Performance Data Mining) (2000).

[22] J. Han, J. Pei, Mining frequent patterns by pattern-growth: methodology and implications, ACM SIGKDD Explorations (special issue on Scalable Data Mining Algorithms), December 2000.

[23] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, M.-C. Hsu, FreeSpan: frequent pattern-projected sequential pattern mining, Proceedings of 2000 International Conference on Knowledge Discovery and Data Mining, August 2000, pp. 355–359.

[24] J. Pei, J. Han, L.V.S. Lakshmanan, Mining frequent itemsets with convertible constraints, Proceedings of 2001 International Conference on Data Engineering, 2001.

[25] J. Han, L.V.S. Lakshmanan, R.T. Ng, Constraint-based, multidimensional data mining, Computer (special issues on Data Mining) (1999) 46–50.

[26] D. Kifer, C. Bucila, J. Gehrke, W. White, DualMiner: a dual-pruning algorithm for itemsets with constraints, in: Proceedings of the 8th ACM SIGKDD International Conference on knowledge discovery and data mining, 2002.

[27] L.V.S. Lakshmanan, R. Ng, J. Han, A. Pang, Optimization of constrained frequent set queries with 2-variable constraints, Proceedings of 1999 ACM-SIGMOD Conference on Management of Data, June 1999, pp. 157–168.

[28] J. Pei, J. Han, Can we push more constraints into frequent pattern mining? Proceedings of 2000 International Conference on Knowledge Discovery and Data Mining, August 2000.

[29] A.K.H. Tung, J. Han, L.V.S. Lakshmanan, R.T. Ng, Constraint-based clustering in large databases, Proceedings of 2001 International Conference on Database Theory, January 2001.

[30] J. Hipp, U. Güntzer, G. Nakhaeizadeh, Algorithms for association rule mining—a general survey and comparison, SIGKDD Explorations 2 (1) (2000) 58–64.

[31] D. Cheung, J. Han, V. Ng, C.Y. Wong, Maintenance of discovered association rules in large databases: an incremental updating technique, Proceedings of 1996 International Conference on Data Engineering, February 1996, pp. 106–114.

[32] D. Cheung, S.D. Lee, B. Kao, A general incremental technique for updating discovered association rules, Proceedings of the International Conference on Database Systems for Advanced Applications, April 1997.

[33] S. Thomas, S. Bodagala, K. Alsabti, S. Ranka, An efficient algorithm for the incremental updation of association rules in large databases, Proceedings of 1997 International Conference on Knowledge Discovery and Data Mining 1997.

[34] N.F. Ayan, A.U. Tansel, E. Arkun, An efficient algorithm to update large itemsets with early pruning, Proceedings of 1999 International Conference on Knowledge Discovery and Data Mining, 1999.

[35] S. Brin, R. Motwani, J.D. Ullman, S. Tsur, Dynamic itemset counting and implication rules for market basket data, ACM SIGMOD Rec. 26 (2) (1997) 255–264.

[36] R. Srikant, R. Agrawal, Mining generalized association rules, Proceedings of the 21st International Conference on Very Large Data Bases, September 1995, pp. 407–419.

[37] H. Toivonen, Sampling large databases for association rules, Proceedings of the 22nd VLDB Conference, September 1996, pp. 134–145.

[38] A. Mueller, Fast sequential and parallel algorithms for association rule mining: a comparison, Technical Report CS-TR-3515, Department of Computer Science, University of Maryland, College Park, MD, 1995.